A Better Role System for OpenMath

Florian Rabe, Michael Kohlhase

Computer Science, Jacobs University Bremen
{f.rabe,m.kohlhase}@jacobs-university.de

Abstract. OpenMath is a standard for the representation and communication of mathematical objects, which are built up from symbols and variables using applications, binding expressions, and key-value attributions. OpenMath2 introduced a set of symbol roles that can be specified in content dictionaries to restrict the occurrences of the respective symbols. This yields a simple, high-level notion of well-formed objects.
While this system is appealing in its simplicity, the definition of well-formedness is purely extensional without an intuitive or formal condition that distinguishes well-formed objects from ill-formed ones. Moreover, some well-formed objects should arguably rather be ill-formed. We try to remedy that with a refined role system while preserving the simplicity of the existing one. In particular, by distinguishing syntactic and semantic roles, we can capture the intuitive notion of well-formedness better.

1 Introduction

OpenMath is a standard for the representation and communication of mathematical objects, which are built up from symbols and variables as applications, binding expressions, attribution. To provide a simple, high-level well-formedness criterion the OpenMath2 standard [BCC$^+$04] introduced a set of symbol roles that can be specified in content dictionaries to restrict the occurrences of the respective symbols. In the rest of this section we point out some problems of the OpenMath Role system before we will try to solve them in the Section 2. Section 4 concludes the paper.

1.1 General Problems of the OpenMath2 Role System

However, even though OpenMath attempts to use the role system for a high-level well-formedness check, its status remains somewhat unclear. The text says that

> *A symbol [...] cannot be used to construct a compound OpenMath object in a way which requires a different role (using the definition of construct given earlier in this section). This means that one cannot use a symbol which binds some variables to construct, say, an application object. [BCC$^+$04, subsection. 2.1.4]*

the compliance chapter [BCC+04, Chapter 5] does not mention the role system at all. Furthermore, even the text above is not consequent enough to forbid clearly ill-formed expressions where binders and keys occur anywhere in an expression or where symbols without any role are used as binders. Even worse: The standard permits composed objects as the first child of a binder; thus, any symbol can be used as a binder after all by wrapping it in a meaningless attribution.

Thirdly, while it is reasonable to avoid a type system in OpenMath, it would be easily possible and extremely helpful to restrict the number of arguments that a symbol can be applied to.

1.2 Complex Binding Operators and Attribution Keys

In a nutshell, the OpenMath role system uses the roles `binder`, `error`, `attribution` and `semantic-attribution`, `application`, and `constant`. The underlying design principle of the OpenMath role system is that binders, errors, keys, and applications occur as the first children of `OMBIND`, `OME`, `OMATP`, and `OMA` objects. While this is certainly appealing, there are several disadvantages.

However, composed expressions must be allowed as applications in order to permit anonymous functions. Therefore, we have the choice to either permit composed binders, errors, and keys or to break the symmetry between the constructors.

Considering the former option, it is indeed often convenient to use binders and keys that are composed expressions, namely the results of applications. For instance we have found that integration can be expressed most elegantly if the integral is an operator that takes the domain of integration as the argument and returns the binder . For example,

```
<OMBIND>
  <OMA>
    <OMS cd="calculs1" name="integral"/>
    <OMA>
      <OMS cd="interval1" name="interval"/>
      <OMV name="a"/>
      <OMV name="b"/>
    </OMA>
  </OMA>
  <OMBVAR><OMV name="x"/></OMBVAR>
  $\boxed{f(x)}$
</OMBIND>
```

is a most natural representation of $\int_a^b f(x)dx$. Here and in the future, we will use boxed mathematical formulae to abbreviate OpenMath Objects wher the XML representations is immaterial to the exposition.

Keys are typically atomic but not always. For example, the typing relation in a language with an infinite type hierarchy is parametrized by an integer value for the type level.

Furthermore, both for binder and for keys, it is conceivable to use symbols wrapped in non-semantic attributions, i.e., to attach presentation information to them.

2 A two-dimensional Role System

Roles are associated with symbols in content dictionaries. We propose to generalize the role of a symbol into two orthogonal aspects:

– `role` represents the syntactic role of a symbol. It corresponds roughly to the `role` of OpenMath2.
– `arguments` represents the number of arguments that a symbol takes.

In this section, we will present the extended role system on a conceptual level and deal with syntactic issues in Section 3.

The `semrole` has three possible values:

– `term`: This role represents all kinds of expressions as they occur in mathematics and type theories. It is the default if no role is given.
– `binder`: This role represents binding operators. There is an informal consensus among mathematicians and computer scientists that expressions and binders are to be distinguished. The characteristic feature of binders is that they need variables and scope and cannot occur alone. For example, in lambda calculus almost everything is an expression but not the λ itself (and not the application operator, which is present in OpenMath already anyway). Similarly, every mathematician would interpret a $\int$ symbol occurring by itself as the non-binding operator on functions and never as a binder.
– `attribution` and `semantic-attribution`: These roles represent keys that can occur in attributions. Just like binders, they do not carry mathematical meaning on their own and only become meaningful within an attribution.

We will abbreviate these four values as $\mathcal{T}$, $\mathcal{B}$, $\mathcal{A}$, and $\mathcal{S}$, respectively.

There is no value `error` because we hold that the property of being an error is not a syntactic property like those of being a binder or a key. Rather, it is a semantic property. This is confirmed by programming languages such as Java or SML where exceptions are treated as normal expressions that only obtain their special semantics in the type-checking and execution phase. Therefore, we argue that `OME` objects should be abandoned in favor of `OMA` objects. The property of being an error should be marked up by introducing a second role attribute for semantic roles. This new attribute is not only useful to mark up errors, but can also be used to mark up other semantic roles such as element, sort, proof, or judgment. We come back to this in Sect. 2.2.

There is no value `application` either. This is because the argument why binders and keys should be separated from expressions does not carry over to applications: A symbol designated as an application may very well occur separately and has a well-defined meaning if it does. For example, in the context of natural numbers, + has the set-theoretical meaning $\{((x, y), z) \mid x + y = z\}$.

We do not consider the property of constructing an application to be alternative to that of constructing a binder or a key. Rather do we consider it as an orthogonal property via the `arguments` attribute.

The attribute `arguments` has as values a natural number or the special value $*$. Its intended semantics is that it gives the number of arguments a symbol takes.

In particular, by making the number of arguments 0 a symbol is forbidden from occurring as the first child of an `OMA` element. We also permit the special value $*$ to make the number of arguments unrestricted. If a symbol has the `semrole` of a binder or key, the default value of `arguments` is 0. This reflects the fact that binders and keys are typically atomic. If the `semrole` is `term`, the default value is $*$.

2.1 Well-formed Objects

We define the well-formed objects E and their syntactic roles $R(E) \in \{\mathcal{T}, \mathcal{B}, \mathcal{A}, \mathcal{S}\}$ in a mutual induction.

1. Every symbol $E = OMS(S)$ is a well-formed expression, and $R(E)$ is the value of the `semrole` attribute of S.
2. Every variable is a well-formed expression with role $\mathcal{T}$.
3. If $E, E_1, \ldots, E_n$ ($n > 0$) are well-formed expressions, $R(E_i) = \mathcal{T}$ for all i, and either
   - E is a composed expression and $R(E) = \mathcal{T}$ or
   - E refers to a symbol and the value of that symbol's `arguments` attribute is $*$ or n,

   then $OMA(E, E_1, \ldots, E_n)$ is well-formed, and its role is $R(E)$.
4. If $E, V_1, \ldots, V_n, E'$ are well-formed expressions, $R(E) = \mathcal{B}$, $R(E') = \mathcal{T}$, and all V_i are well-formed attributed variables and $R(V_i) = \mathcal{T}$, then $OMBIND(E, (V_1, \ldots, V_n), E')$ is a well-formed expression with role $\mathcal{T}$.
5. If E, K, E' are well-formed expressions, $R(E') = \mathcal{T}$, and $R(K) \in \{\mathcal{S}, \mathcal{A}\}$, then $OMATTR(E, K := E')$ is a well-formed expression with role $R(E)$. (For simplicity, we omit the analogous case of multiple attributions.)
6. All elements of specific domains (numbers, strings) and all foreign objects are well-formed expression with role $\mathcal{T}$.

Note that we again omit the case of error objects. We come back to them in Sect. 2.2.

It is simple to make a RelaxNG schema out of the above definition. The schema can be generated from the CDs as in [Koh08].

Our role system satisfies the following invariants:

- Only terms may occur as arguments in applications, variables or scopes in bindings, or values in attributions.
- Keys and binders can only occur as the heads of attributions and bindings, respectively, and nothing else can occur in these positions.
- All symbols can take arguments, and the role of the symbol determines the role of the result. Symbols can be prevented from taking arguments and thus from occurring as the head of an application by making their number of arguments 0.
- All expressions can be attributed, and attributions do not change the role of the attributed expression.

In Case 5, it may be reasonable to additionally require $R(E) = \mathcal{T}$, which has the effect that only terms may be attributed. While it seems reasonable to permit attributions to binder or keys, this might be sacrificed for backwards compatibility (see below). (*)

2.2 Semantic Roles

In addition to distinguishing syntactic roles, it is often useful to give symbols with syntactic role $\mathcal{T}$ an additional semantic role attribute `semrole`. The intuition behind this becomes clear from the following list of possible values.

- `element` and `sort`: These roles represent mathematical objects and their containers.
- `proof` and `judgment`: Following the Curry-Howard correspondence, these roles represent proof terms and their containers, namely judgments about mathematical objects.
- `error` and `error-type`: These roles represent error objects and their containers.

In order to define the semantic role of an arbitrary term, we additionally permit the `semrole` attribute on an OMV element when occurring within an `OMBVAR` element. This is useful to give variables a semantic role. For example to distinguish between a lambda abstraction over elements or over types.

Both on variables and symbols, `element` is the default if no semantic role is given.

The semantic role of a well-formed expression with syntactic role $\mathcal{T}$ is defined as follows:

1. For a symbol: according to the declaration of the symbol.
2. For a variable: according to the declaration of the variable.
3. The semantic role of an application is that of the first child.
4. The semantic role of a binding is that of the third child.
5. The semantic role of an attribution is that of the attributed expression.
6. The semantic role of an element from a specific domain or foreign object is `element`.

Then the OME objects of OpenMath2 can be recovered as syntactic sugar for OMA objects with semantic role `error`.

2.3 Conservativity and Backwards Compatibility

The current roles of the OpenMath2 standard can be translated to ours as follows:

- `constant`: role $\mathcal{T}$ with 0 arguments,
- `application`: role $\mathcal{T}$ with $*$ arguments,
- `error`: role $\mathcal{T}$ with $*$ arguments and semantic role `error`.

– `binder`: role $\mathcal{B}$ with 0 arguments,
– `attribution`: role $\mathcal{A}$ with 0 arguments,
– `semantic-attribution`: role $\mathcal{S}$ with 0 arguments,

Under this translation, we can prove the following restrictions for well-formed expressions in our sense over content dictionaries in the OpenMath2 sense:

– `constant`: This symbol may only occur by itself or with attributions: on toplevel, as an argument of an application, as the scope of a binding, or as a value of an attribution.
– `application` or `error`: This symbol may only occur as the first child of an application (possibly with attributions) or anywhere where symbols with role `constant` can occur.
– `binder`: This symbol may only occur as the first child of a binding (except that in that position it may have attributions)$^{+}$.
– `attribution` or `semantic-attribution`: This symbol may only occur as the key of an attribution (except that in that position it have attributions)$^{+}$.

Here the caveats marked with $^{+}$ disappear if we adopt the variant of the definition of well-formed objects marked (*) in Sect. 2.1.

This means that OpenMath2 content dictionaries can be translated to our role system in a way that well-formed expressions in our sense come as close to the apparently intended meaning of the OpenMath2 standard as possible.

3 Proposed Changes to the OpenMath Standard

We propose the following changes to the OpenMath standard.

1. Symbols declarations in content dictionaries have three attributes
   – `role`, values: `term` (default), `attribution`, `semantic-attribution`, `binder`,
   – `arguments`, values: natural numbers or *, * is default if `role` is `term`, 0 is default otherwise,
   – `semrole`, values: `element` (default), `sort`, `proof`, `judgment`, `error`, `error-type`.
2. OMV elements in variable declarations have a `semrole` attribute as above.
3. OMATP elements may have arbitrary objects as the first child.
4. The definitions of well-formed object, syntactic role of an object, and semantic role of a term from Sect. 2.1 are added to the standard and replace the existing descriptions of well-formed objects.
5. OME elements become syntactic sugar for OMA element where the first child has semantic role `error`.

4 Conclusion

In this paper we have critically re-accessed the role system introduced in the OpenMath 2 standard. While this system is appealing in its simplicity, it has several drawbacks that we try to solve in this paper by generalizing roles into independent syntactic and semantic flavors.

This has the benefit that a straightforward and formal definition of well-formed objects can be achieved that preserves the generality and simplicity of OpenMath 2 while ruling out many so far permitted nonsensical objects. By adding the possibility of restricting the number of arguments of a symbol, users are able to succinctly restrict the possible uses of a symbol without incurring a significant gain in complexity.

Users can exploit our role system to characterize the possible first children of composed expressions more strictly as before, and these restrictions lead to invariants that are available to applications. Our role system would also tremendously simplify our definition of a set-theoretic semantics of OpenMath ([**?**]), which currently has to go out of its way to interpret practically useless objects.

Our extension is conservative in the sense that existing content dictionaries can be translated to our proposed system. Formerly well-formed objects stay well-formed except for those cases which the OpenMath2 role system – accidentally in our opinion – permitted.

References

BCC+04. Stephen Buswell, Olga Caprotti, David P. Carlisle, Michael C. Dewar, Marc Gaetano, and Michael Kohlhase. The Open Math standard, version 2.0. Technical report, The Open Math Society, 2004.

Dav99. James Davenport. A small OpenMath type system. Technical report, The OpenMath Esprit Project, 1999.

Koh08. Michael Kohlhase. Compiling OpenMath type systems to Relax NG grammars. In Olga Caprotti, Sebastian Xambó, Maria-Antonia Huertas, Michael Kohlhase, and Mika Seppälä, editors, *3rd JEM Workshop – Joining Educational Mathematics*, 2008.